BANCO DE DADOS AVANÇADO

AULA 02

ENTIDADE, ATRIBUTO E RELACIONAMENTO
ENTIDADE:
Representa um objeto do mundo real que precisa ser armazenado no banco de dados.
Cada entidade pode ter várias ocorrências
Pessoa pôde representar clientes, funcionários, alunos,etc.

Produto pode ser um item de um e-commerce.

Pedido pode representar uma compra realizada.

ID	NOME	E-mail	Data Nascimento
1	João Silva	joão@email.com	15/06/1990
2	Maria Lira	maria@email.com	23/09/1985

ATRIBUTOS
Características ou propriedades de uma entidade.
Tipos de atributos:

Atributo simples: não pode ser dividido (exemplo CPF, nome).

Atributos compostos: podem ser divididos em partes menores (exemplo nome completo pode ter "primeiro nome " e "sobrenome").

Atributos derivados: podem ser calculado com base em outro atributo (exemplo idade pode ser calculada a partir da data de nascimento).
EXEMPLO DE ATRIBUTO DA ENTIDADE CLIENTE:

- ID (CHAVE PRIMÁRIA)
- NOME (SIMPLES)
- ENDEREÇO (COMPOSTO: RUA, NÚMERO, CIDADE E ESTADO)
- TELEFONE (MULTI VALORADO PORQUE A PESSOA PODE TER MAIS DE UM TELEFONE)
- IDADE (DERIVADO DA DATA DE NASCIMENTO)

RELACIONAMENTO:
TIPOS DE RELACIONAMENTO:

1) UM PARA UM (1:1) um cliente tem um único CPF.

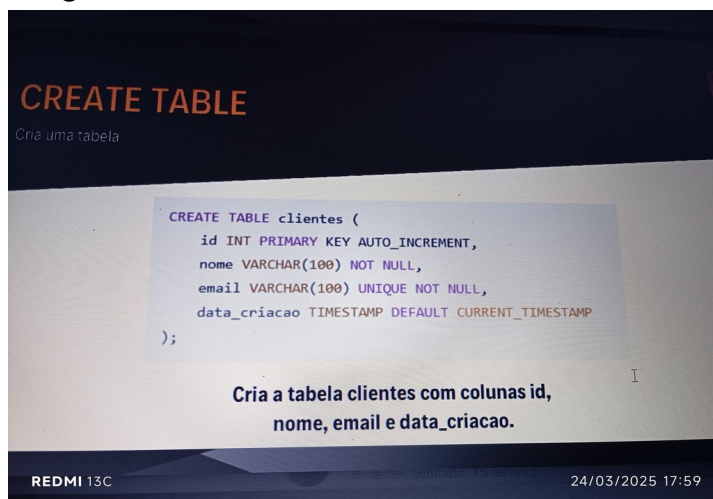
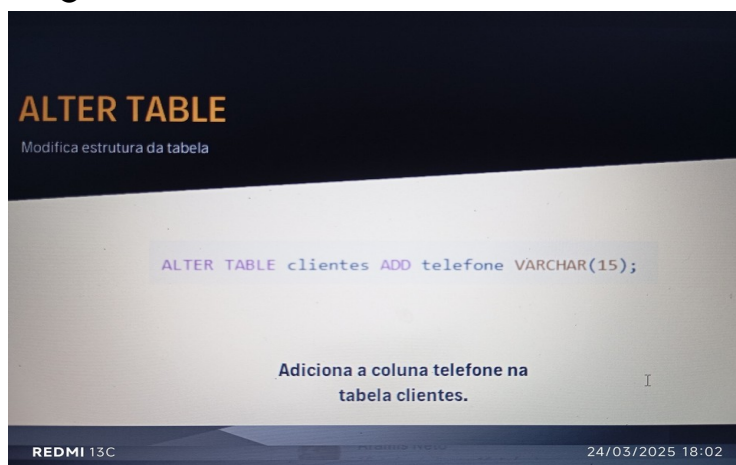
2) UM PARA MUITO (1:N) um cliente pode fazer vários pedidos.
3) MUITOS PARA MUITOS (M:N) um aluno pode estar matriculado em vários cursos, e um curso pode ter vários alunos.

- Entidade cliente (ID, nome e e-mail)
- Entidade pedido (ID, data, cliente_ID)

INTRODUÇÃO AO SQL:
DDL E DML

SQL
- Structured Query Language (SQL)
- Linguagem padrão para banco de dados relacionais
- Utilizada para criação manipulação e consultas de dados

DDL
Criação e estruturação
- Data definition language

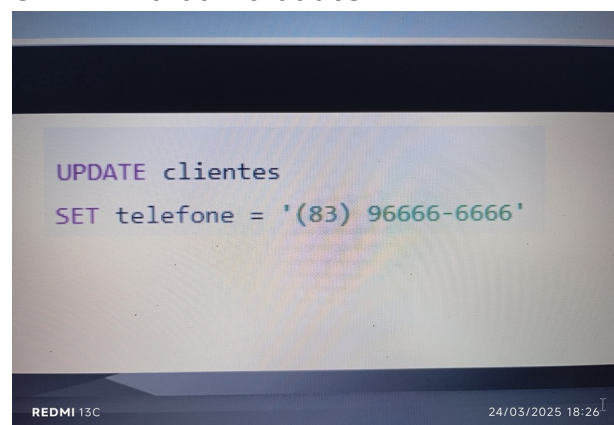
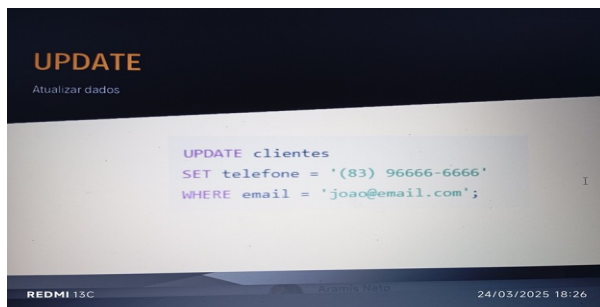
Permite definir estruturas de dados.
- CLIENTE cria a tabela cliente com colunas id, nome, e-mail e data criação

- ALTER modifica estrutura da tabela
- Adiciona a coluna telefone na tabela cliente.

DROP exclui uma tabela
Remove completamente a tabela cliente e seus dados
DROP TABLE clientes;

REMOVER UMA COLUNA
ALTER TABLE clientes DROP COLUMN celular;
Remove a coluna celular.

DML
manipulação de dados
Permita a manipulação de dados em um banco de dados
INSERT e sério novo cliente com nome, e-mail e telefone na tabela cliente.
INSERT INTO clientes (nome, email, telefone)
VALUES (“ JOÃO SILVA”, “JOÃO@GMAIL.COM” , (83)99999-9999);
Insere vários registros de uma só vez

SELECT: faz consulta de dados
Retorna todos os clientes cadastrados.
Exemplo:
SELECT nome, email FROM clientes;
Retorna apenas os nomes e email dos clientes.
Usando filtro com WHERE
SELECT * FROM clientes WHERE email = ‘joão@gmail.com’;

Ordenando o resultado
SELECT * FROM clientes ORDER BY nome ASC;
ASC em ordem crescente
DESC em ordem decrescente

UPDATE atualiza dados

DELETE remove os dados
Delete FROM clientes WHERE email = ‘ joão@gmail.com’;

AULA 03
NORMALIZAÇÃO DE DADOS
A normalização de dados em banco de dados é um processo de organizar os dados de forma eficiente para minimizar redundâncias e dependências garantindo que os dados sejam armazenados de maneira estruturada e consistente.

OBJETIVOS

- Evitar erros
- Garante a consistência e integridade dos dados
- Diminui a redundância
- Facilita a gestão e a tomada de decisões
- Simplifica a manutenção
- Melhor é o desempenho de consulta

COMO ELA É FEITA

- Criado tabelas
- Estabelecendo relações entre as tabelas
- Dividindo os dados em tabelas menores
- Interligando as tabelas de forma lógica
- Assegurando que cada dado seja armazenado apenas uma vez

Exemplo
Imagino a tabela de cadastro de alunos e cursos. Se armazenássemos todas as informações do aluno e seu curso em uma única linha haveria repetições de dados.

FORMAS NORMAIS
Cada forma normal tem um conjunto de critérios que deve ser atendido para reduzir problemas como publicações de dados e anomalias de seção atualização e execução.

PRIMEIRA FORMA NORMAL
Elimina grupos repetitivos

- A tabela deve ter valores indivisíveis.
- SEGUNDA FORMA NORMAL Não pode haver colunas com múltiplos valores ou listas.
- Cada célula deve conter apenas um valor.

Em uma tabela de cliente, ao invés de ter uma única coluna para endereço, seria melhor dividir em colunas separadas para a rua, cidade, estado e Código postal.

TABELA NÃO NORMALIZADA

ID	Nome	Curso
1	João	Matemática,Física
2	Maria	Química,Biologia

ELIMINAR GRUPOS REPETITIVOS
TABELA NORMALIZADA

ID	Nome	Curso
1	João	Matemática
1	João	Física
2	Maria	Química
3	Maria	Biologia

SEGUNDA FORMA NORMAL:

- A tabela deve estar na primeira forma normal
- Todas as colunas não não chave devem depender totalmente da chave primária.
- Isso é necessário quando temos Chaves composta (quando a chave primária é formada por mais de uma coluna).

Segundo forma normal eliminando dependências parciais
CHAVE COMPOSTA É UMA COMBINAÇÃO DE DUAS OU MAIS COLUNAS DE UMA TABELA QUE IDENTIFICAM EXCLUSIVAMENTE CADA REGISTRO.

Funcionário
Matricula:int chave primária
cpf:int chave composta
nome; Varchar
salario: double
função; varchar
Fk_cod_departamento:int chave estrangeira

Departamento

cod_departamento: int
nome:varchar

ELIMINAR AS DEPENDÊNCIAS PARCIAIS

ID	Nome	Curso	Professor
1	João	Matemática	Ana
1	João	Física	Carlos
2	Maria	Biologia	José

TABELA NÃO NORMALIZADA(VIOLANDO A 2FN)
A coluna professor depende apenas da coluna curso e não da chave composta (ID + Curso).
SEGUNDA FORMA NORMAL
TABELA 1 ALUNOS

ID	Nome
1	João
2	Maria

TABELA 2 CURSO

Curso	Professor
Matemática	Ana
Física	Carlos
Biologia	José

TABELA 3 MATRÍCULA

ID	Curso
1	Matemática
2	Física
3	Biologia

A tabela curso armazena apenas as informações dos professores, enquanto a tabela matrícula relaciona os alunos aos cursos.

TERCEIRA FORMA NORMAL
Eliminar dependências transitivas

- A tabela deve estar na segunda forma normal

- Nenhuma coluna não não chave deve depender de outra coluna não chave (dependência transitiva).

Tabela não normalizada (violando a terceira forma normal).

ID	Nome	Curso	Professor	Sala
1	João	Matemática	Ana	A101
1	João	Física	Carlos	A102
2	Maria	Biologia	Jose	A103

A coluna sala depende de professor, que é uma coluna não chave, que é uma dependência transitiva.

AULA 04
SUBCONSULTAS

- É uma consulta SQL inserida dentro de outra consulta principal, elas são usadas para recuperar dados que serão utilizados na consulta externa.
- Filtrar resultados com valores obtidos dinamicamente comparações avançadas (exemplo: encontrar funcionários com salário acima da média) uso dentro de SELECT , WHERE,HAVING,FROM,etc.

SUBCONSULTAS ESCALAR (RETORNA UM ÚNICO VALOR)
Usada quando a subconsulta retorna apenas um único valor (uma linha de uma coluna).
Exemplo: encontrar funcionário com salário acima da média

```
SELECT nome, salario
FROM funcionários
WHERE salário > ( SELECT ( salario) FROM funcionarios);
```

A subconsulta (SELECT AVG (salário) FROM funcionários) retorna um valor (a média dos salários), e é usado na condição WHERE.

SUBCONSULTA DE MÚLTIPLAS LINHAS (RETORNA MAIS DE UM VALOR)

Usada quando subconsulta retorna mais de uma linha e precisa ser combinada com operadores como IN , ANY , ALL.

Exemplo: encontrar produto que estão impedidos

```
SELECT nome_produto
FROM produtos
WHERE id_produto IN ( SELECT id_produto FROM pedidos);
```

A subconsulta retorna uma lista de ID_produto que aparecem na tabela pedidos. O IN compara se o ID_produto da tabela produtos está nesta lista.

SUBCONSULTA CORRELACIONADA (EXECUTADA PARA CADA LINHA DA CONSULTA EXTERNA)

Diferente das anteriores, essa subconsulta depende dos valores de consulta externa e executada repetidamente para cada linha.

Exemplo: encontrar funcionário que ganha mais do que a média do seu próprio departamento:

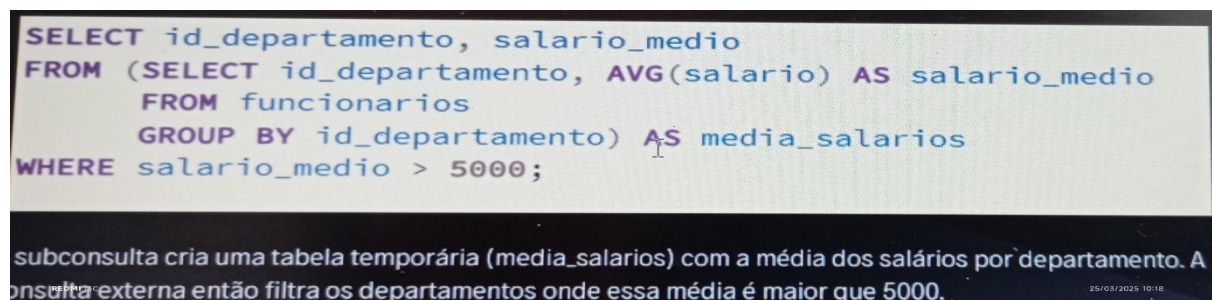
```
SELECT nome, salário, id_departamento
FROM funcionários F1
WHERE salário > ( SELECT AVG( salário) FROM funcionários F2
WHERE F1.id_departamento = F2.id_departamento);
```

Para cada funcionário (F1) subconsulta calcula a média salarial do seu próprio departamento (F2).

SUBCONSULTA NO FROM (USADA COMO TABELA DERIVADA)

Uma subconsulta pode ser usada dentro do FROM funcionario como uma tabela temporária.

```
SELECT id_departamento, salario_medio
FROM (SELECT id_departamento, AVG(salario) AS salario_medio
      FROM funcionarios
      GROUP BY id_departamento) AS media_salarios
WHERE salario_medio > 5000;
```

subconsulta cria uma tabela temporária (media_salarios) com a média dos salários por departamento. A
onsulta externa então filtra os departamentos onde essa média é maior que 5000.

A subconsulta cria uma tabela temporária (média_salário) com a média do salário por departamento. A consulta externa então filtra os departamentos onde essa média é maior que 5.000.

RELEMBRANDO OS CONCEITOS

1. WHERE filtrar dados

WHERE é usado para filtrar e registrar em uma consulta. ele define condições para selecionar apenas os dados que atendem a certos critérios.

```
SELECT nome, salario
FROM funcionários
WHERE salario > 5000;
```

2. AVG() - calculando médias

A função AVG() retorna a média de um conjunto de valores

```
SELECT AVG( salario) AS medias_salario
FROM funcionários;
```

Usamos AS para dar o nome mais amigável ou descritiva a uma coluna.

```
SELECT nome AS Nome_Fucionario, salario_Atual
FROM funcionários
```

```
SELECT f.nome, d.nome_departamento
FROM funcionarios f
JOIN departamentos d ON f.id_departamento = d.id_departamento;
```

REDMI 13C 25/03/2025 10:38

3. IN - verificando se um valor está em uma lista

O operador IN verifica se um valor está dentro de uma lista retornada por uma subconsulta ou definida manualmente.

EXEMPLO SELECIONAR PRODUTO QUE ESTÃO IMPEDIDOS

A SUBCONSULTA RETORNA À LISTA DE ID PRODUTO QUE APARECEM NA TABELA PEDIDOS, E O IN VERIFICA SE CADA PRODUTO DA TABELA PRODUTOS ESTÁ NESSA LISTA.

4. GROUP BY - agrupa os dados

O GROUP BY agrupa linha com o mesmo valor em uma coluna e permite usar funções de agregação (AVG , SUM, COUNT , etc).

Exemplo:

SELECT id_departamento, AVG (salario) AS salário _media
FROM funcionários
GROUP BY id_departamento;

- O GROUP BY id_departamento agrupa os funcionários por departamento.
- A função AVG (SALÁRIO) é calculada para cada grupo (departamento).

AULA 05

JUNÇÕES (JOINS)

Junção são operações fundamentais em banco de dados relacionais que permite combinar registro de duas ou mais tabelas com base em colunas comuns. Elas são usadas para recuperar dados distribuídos em diferentes tabelas e facilita análises complexas

POR QUE USAR JUNÇÕES?

- EVITA REDUNDÂNCIA DE DADOS.
- MELHOR A ORGANIZAÇÃO E A NORMALIZAÇÃO DO BANCO DE DADOS.
- FACILITA A RECUPERAÇÃO E EFICIÊNCIA DE INFORMAÇÕES RELACIONADAS.

CONSTRUINDO A DECLARAÇÃO

CLÁUSULA ON

Determine a condição de JOIN, quem indica com mais tabelas deve ser comparadas

No geral, a comparação ocorre por meio de um relacionamento entre chave primária na primeira tabela e chave estrangeira na segunda tabela

CONDIÇÃO DE JOIN

Nomeia uma coluna em cada tabela envolvida no JOIN e indica como as colunas devem ser comparadas

No geral, usamos operador = para obter linhas como colunas correspondentes. É como usar o relacionamento de PK de uma tabela com FK de outra tabela

NOMES DE COLUNAS QUALIFICADA

Nome da coluna precedido pelo nome da tabela a qual pertence, separado por um ponto (.)

Usamos nomes de colunas qualificadas para identificar a qual tabela cada campo envolvido pertence, evitando erro de ambiguidade caso uma coluna tenha o mesmo nome em duas tabelas diferentes

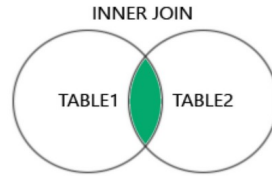
ON pedidos _codigo_produto = produtos.codigo_produto

SINTAXE INNER JOIN

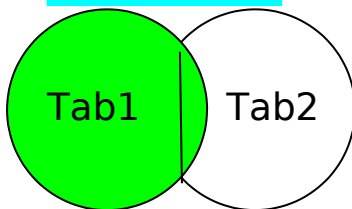
```
SELECT colunas
FROM tabelas 1
INNER JOIN tabela2
ON tabela1.coluna = tabela 2.coluna
```

TIPO DE JUNÇÕES BÁSICAS

INNER JOIN

Retorna apenas os registros que possuem
correspondência em ambas as tabelas. É o tipo mais comum de junção e é usado para buscar informações relacionadas diretamente.

```
SELECT clientes.nome, pedidos.id_pedido
FROM clientes
INNER JOIN pedidos ON clientes.id_clientes = pedidos.id_clientes;
```

LEFT JOIN (OUTER LEFT JOIN)

Retorna todos os registros da tabela da esquerda e os recorrespondente da direita. Quando não há correspondência o resultado exibe Null para os campos da tabela da direita.

```
SELECT clientes.nome, pedidos.id_pedido
FROM clientes
LEFT JOIN pedidos ON clientes.id_clientes,id_clientes;
```

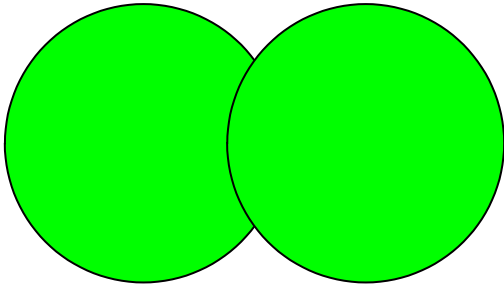
RIGHT JOIN (OUTER RIGHT JOIN)

Funciona de maneira semelhante ao Left Join, mas retorna todos os registros da tabela da direita e os correspondentes da esquerda. Quando não há correspondência retorna Null para os campos da tabela da esquerda.

```
SELECT clientes.nome, pedidos.id_pedido
FROM clientes
RIGHT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

FULL OUTER JOIN

Retorna todos os registros de ambas as tabelas preenchendo com Null onde não há correspondência

FULL OUTER JOIN

```
Select calientes.nome, pedidos.id_pedido
FROM clientes
Full JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```